Exkurs: Messung der Laufzeit am Arduino

Dienstag, 11. Oktober 2022 10:39

Wie lange dauert es einen char, int, long, long long in der AVR Architektur zu addieren?

Exkurs: Zeitmessung

Themen: ungewollte Compiler Optimierungen
Zeitmessung mit AVR

millis - Arduino Reference
micros - Arduino Reference

Achtung! Die halbwegs aussagekräftige Zeitmessung auf einem Computer (egal ob PC oder Microcontroller) ist eine Wissenschaft für sich. Ergebniss unbedingt hinterfragen!

Der ATmega328p hat eine Taktfrequenz von 16MHz
Die allereinfachsten Rechenoperationen auf einer CPU benötigen genau einen Arbeitstakt

→ Lösung suche aus dem ATmega328p Befehlssatz einen Befehl, der nur einen Takt benötigt (Takt engl: cycles)
→ Wie oft kann dieser Befehl auf dem ATmega328p in einer Sekunde ausgeführt werden?

z.B ADC - Add with carry
ein ADC braucht $T = \frac{1}{f}$ d.h. $6{,}25 \cdot 10^{-8}$ s. Also können 16 Millionen ADC in der Sekunde ausgeführt werden
16 Takte in 1 µs

Time: Auflösung: 4µs → 64 Takte
1 Arbeitstakt: 16MHz → 0,0625µs

→ Wieviele Arbeitstakte sind vergangen, wenn sich micros() um 1 erhöht hat?
→ Versuchen Sie mithilfe von micros() festzustellen, wie lange auf dem ATmega328p unsigned char + unsigned char dauert.

```
void setup()
{
  Serial.begin(9600);
}

void loop()
{
  unsigned char a = 0x7;
  unsigned char b = 0xff;
  long time_start = micros();
  b = b + a;
  long time_stop = micros();
  delay(10);
  Serial.print("Benoetigte Zeit: ");
  Serial.println(time_stop - time_start);
}
```

b = 0x106 262 Dezimal

Achtung:
time_start könnte 0xff sein und time_stop 0x02
Dies ist aber kein Problem siehe Arithmetik mit Ganzzahlen (Subtraktion)

→ Warum wird oft 0µs angezeigt? Löschen Sie die Rechenoperation und vergleichen Sie die Ergebnisse! Was ist da los?

→ Ändert sich etwas, wenn wir b wirklich benutzen? z.B. Serial.println(b);

Der C-Compiler merkt, dass wir zwar b+a rechnen, dass Ergebnis aber nicht verwendet wird. Deshalb löscht der Compiler die Zeile b = b + a; weil sich nichts bewirkt.
Vorsicht: Man könnte meinen durch ein Serial.println(b); muss der Compiler b = b + a; ausführen. Nein, das wird er nicht tun, sondern während der Compilezeit ausrechnen, was a+b ist, da der Compiler versteht, dass a und b feste Zahlen sind. Er wird letztlich so tun als hätten wir geschrieben Serial.println("262")

Ergebnis:
Der Compiler ist ein verdammt komplexes Programm, dass sich extrem bemüht unseren geschriebenen Code zu optimieren. Bei der Zeitmessung und allgemein beim Timing bereiten uns aber die Optimierungen des Compilers riesige Kopfschmerzen!

Lösung für die Probleme:
① Die gemessenen Zeiten sind grundsätzlich sehr klein.
Viel öfters addieren, z.B 1000 Mal.
② Der Compiler optimiert die zu messende Addition weg.
Die Variablen a und b als volatile definieren. volatile - Arduino Reference

```
/** Zeitmessung zweiten Anlauf */
```

(2) Der Compiler optimiert die zu messende Addition weg.

Die Variablen a und b als volatile definieren. volatile - Arduino Reference

```
/** Zeitmessung zweiter Anlauf */

void setup()
{
  Serial.begin(9600);
}

void loop()
{
  volatile unsigned char a = 0x7;
  volatile unsigned char b = 0xff;
  long time_start = micros();
  for (int i = 0; i < 1000; i++)
  {
    b = b + a;
  }
  long time_stop = micros();
  delay(10);
  Serial.print("Benoetigte Zeit: ");
  Serial.println(time_stop - time_start);
}
```

Das ist schon nicht schlecht. Wir müssen natürlich Bedenken, dass nun die Gesamtzeit: Schleife, Variable SRAM → CPU Register, Addition, Variable CPU Register → SRAM gemessen wird. TODO? TODO?

Aber: Warum schwanken die Zeiten überhaupt noch?

Wegen Interrupts? Interrupts sind Routinen (C-Funktionen), die der Microcontroller immer wieder außerplanmässig abarbeitet. Dazu unterbricht er unseren Code, führt dann die Interrupt-Routine aus und führt anschließend wieder unser Programm aus. Übrigens werden wir in ITCT eigene Interrupt-Routinen schreiben.

Lösung: mit den folgenden zwei Funktionen können Interrupts verboten/erlaubt werden. Dies sollte nur kurzzeitig geschehen, da Interrupt-Routinen wichtige Aufgaben erledigen.

noInterrupts - Arduino Referenz

interrupts - Arduino Referenz

```
void setup()
{
  Serial.begin(9600);
  noInterrupts();
}
void loop()
{
  volatile unsigned char a = 0x7;
  volatile unsigned char b = 0xff;
  noInterrupts();
  long time_start = micros();
  for (int i = 0; i < 1000; i++)
  {
    b = b + a;
  }
  long time_stop = micros();
  interrupts();
  delay(10);
  Serial.print("Benoetigte Zeit: ");
  Serial.println(time_stop - time_start);
}
```

Timing with Macros

```
/** Zeitmessung: jetzt fangen wir an zu spinnen ... */
/* Durch timer2 gibt es einen overhead, der natürlich selber clockcycles
 * frisst. Evtl könnte man versuchen diesen overhead durch inlinen von
 * timer2.reset() und timer2.get_count() noch weiter zu reduzieren.
 * Aber diesen overhead kann man sowieso herausrechnen indem man die
 * Zeit 8 mal b += a mit der Zeit 16 mal b += a vergleicht!
 *
 * Ergebnisse:
 * Ein b += a braucht:
 * unsigned char:      7/8 * 0.5us -> 7  clockcycle
 * unsigned int:       7/4 * 0.5us -> 14 clockcycle
 * unsigned long:      7/2 * 0.5us -> 28 clockcycle
 * unsigned long long: 9   * 0.5us -> 72 clockcycle
 *
 * 1 clockcycle entspricht 0.0625us
 */

#define MEASURE_START   noInterrupts();     \
                        TCCR1A = 0;         \
                        TCCR1B = 1 << CS10; \
                        TCNT1 = 0;
/** offset empty loop is 4 cycles with optimization off */
/** offset empty loop is 1 cycles with standard optimization */
#define MEASURE_STOP(c, offset)  c = TCNT1;         \
                         c = c - offset;         \
                         interrupts();


typedef unsigned char integer_to_test;
void setup()
{
  Serial.begin(9600);
}

//#pragma GCC push_options
//#pragma GCC optimize ("-O0")
void loop()
{
  volatile integer_to_test a = 0x7;
  volatile integer_to_test b = 0xff;
  unsigned int cycles;
```

```
  MEASURE_START

  b = b + a + a;

  MEASURE_STOP(cycles, 1)
  delay(10);
  Serial.print("Benoetigte Zeit: ");
  Serial.println(cycles);
}
//#pragma GCC pop_options
```